Deep Learning Hyper-parameter Optimization for Video Analytics in Clouds

Muhammad Usman Yaseen, Ashiq Anjum, Omer Rana and Nikolaos Antonopoulos

***Abstract*—A system to perform video analytics is proposed using a dynamically tuned convolutional network. Videos are fetched from cloud storage, pre-processed and a model for supporting classification is developed on these video streams using cloud-based infrastructure. A key focus in this work is on tuning hyper-parameters associated with the deep learning algorithm used to construct the model. We further propose an automatic video object classification pipeline to validate the system. The mathematical model used to support hyper-parameter tuning improves performance of the proposed pipeline, and outcomes of various parameters on system's performance is compared. Subsequently, the parameters that contribute towards the most optimal performance are selected for the video object classification pipeline. Our experiment-based validation reveals an accuracy and precision of 97% and 96% respectively. The system proved to be scalable, robust and customizable for a variety of different applications.**

***Index Terms*—Video Analytics, Cloud Computing, Automatic Object Classification, Deep Learning**

I. Introduction

VIDEO analytics plays a vital role in detecting and tracking temporal and spatial events in video streams. A number of pre-installed cameras, as shown in figure 1, produces video data. This data needs processing to generate useful clusters such as classification and tracking of a marked person. As shown in figure 2, video data captured from different cameras can be used to locate a person of interest. The mapping of the person is then associated with particular locations visited along with the time spent at each location. The large amounts of data makes it nearly impossible for human operators to manually process this data.

Deep learning based Video analytics systems can involve many hyper-parameters, including learning rate, activation function and weight parameter initialization. A trial-and-error approach is mostly followed in selecting these parameters, which makes it time consuming and at times may provide inaccurate results.

To overcome these challenges, we present a system for object classification from multiple videos. We propose hyper-parameter tuning through a mathematical model to achieve higher object classification accuracy. The mathematical model aids in observing the hyper-parameter outcomes on overall performance of the learned model. Values of the hyper-parameters are dynamically varied and appropriate parameters are selected.

We have first performed object extraction which are then scaled and normalized. Each video frame is scaled at a size of 150×150. During our experiments, it was observed that deep learning networks perform better when input data is provided in the normalized form.

The system performs training of the model on multiple distributed processors by utilizing cloud infrastructure. Multiple cloud nodes are used for partial model training. The results from each partial model are then collected at the master. This results in the reduction of the overall training time. The selection of appropriate normalization scheme with gradient descent approach and learning rate helps to move the model score of the system towards stability during training.

We have adapted iterative reduce, an extended form of map-reduce paradigm to perform training quickly and efficiently. The parallel and distributed training also process data rapidly. The apache spark cluster is tuned for maximum resource utilization. The proposed system is customizable in terms of scalability i.e. nodes can be added or removed with the addition or deletion of videos.

The evaluation of system is performed on a 100GB video dataset. We present a video object classification pipeline to evaluate the proposed system in which objects of interest are located. We have adopted the techniques from data augmentation including rotation, flip and skew for training due to the limited labeled data for application pipeline. More training data leads to higher accuracy for the classifier by reducing over-fitting and exposing the network to more training samples. Another advantage of applying these transformations is that they make the classifier invariant to typical transformations in the target object which is being located in the video streams.

We have shown that the proposed system performs object classification with high accuracy and we demonstrate experimentally that the distributed training with iterative reduce for automatic video analytics is a promising way of speeding up the training process. After training, the classifier can be stored locally and uses a match probability to classify objects.

There are mainly three contributions in the paper. (i) we devised a mathematical model to observe the outcomes of various hyper-parameter values on system performance. A comparison of different hyper-parameter values has been made and the parameters which give the most optimal performance are selected; (ii) we scaled and configured the (Apache Spark) cluster for parallel model training; (iii) we propose an automatic object classification pipeline to support large scale object classification in video data.

The organization of the paper is as follows: Section II details the related work. Section III explains approach used in carrying out video analysis, using a CNN and hyper-parameter tuning for such a network. Section IV explains the architecture

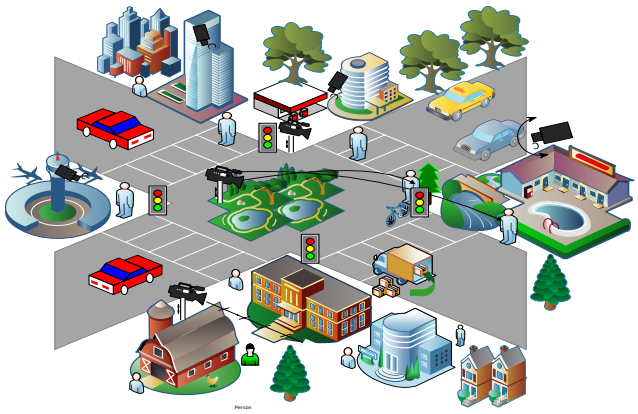

Fig. 1: Video Capture Infrastructure

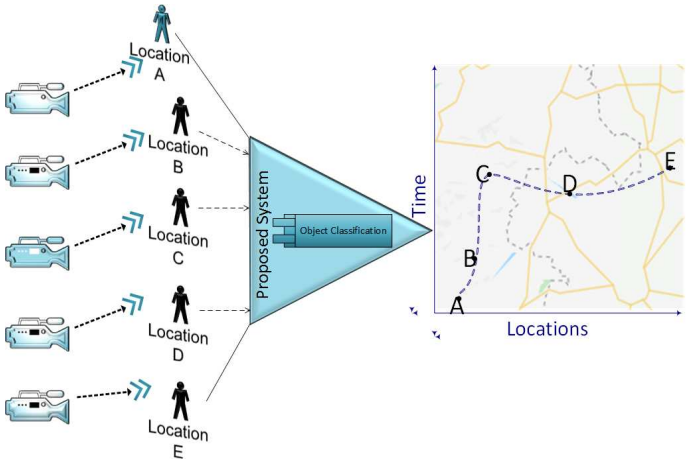

Fig. 2: Mapping of Marked Person

and implementation used to realise our proposed system. Section V describes experimental setup. Results and conclusions are provided in Section VI and Section VII respectively.

II. Related Work

Recent video analytics systems often use shallow networks and hand crafted features to perform object classification[30][31]. These hand crafted features are combined to generate larger features. These larger features provide an estimate of appearance and motion information of objects in the video. These larger features are not suitable for object classification from large video data. [1] proposed a system using GPUs to reduce the computational complexity involved in video stream decoding and processing. An operator could specify the video file and search criteria to a client program, video analytics is then performed on the cloud and results are returned back to the operator after some time. However, this work also involved the use of a shallow (learning) network and produced high dimensional feature vectors. Deep learning networks have emerged as influential tools for solving complex problems such as medical imaging, speech recognition, classification and recognition of objects [23][24][25][26][32]. These networks are capable to perform classification and recognition on large scale data as compared to shallow networks but require more computational resources for training. It also poses many other challenging tasks like hyper-parameter tuning and increasing times for training.

Hyper-parameter optimization has been an area of discussion over the years [17] and mainly included racing algorithms [18] and gradient search [19]. It is now shown that random search is better as compared to grid search. The Bayesian optimization methods can perform even better than random or grid search. Some of the researchers also proposed methods to perform automatic hyper-parameter optimization. The most common implementations of Automatic Bayesian optimization are Spearmint [20], which uses a Gaussian process model (GP) [21]. Also, Tree Parzen Estimator (TPE) [2], which generates a density estimate of each hyper-parameter. These methods have shown competitive results but their acceptance is hampered because of high computational requirements and performs best for problems with few numerical hyper-parameters. On the other hand, the hyper-parameter optimization done manually by human operators is less resource intensive and consumes less time as compared to automated methods. The evaluation of a poor hyper-parameter setting can be quickly detected by human operators after a few steps of the stochastic gradient descent algorithm. They can quickly judge that network is performing bad and can terminate the evaluation.

A number of convolutional neural network models have been proposed in the recent past. Szegedy et al. [3] proposed to modify the CNN by changing the end layer of the network with regression. This modification resulted in the average precision of 0:305 over 20 classes. As opposed to Szegedy's proposed model, Girshick et al. [4] adopted a bottom-up region based deep model called R-CNN. The proposed model generated two thousand region proposals and a CNN was used for feature extraction from each region which were then classified by SVMs. An improvement of 30% in accuracy was observed but it was slow as training was a multi-stage pipeline. Ross Girshick further improved their method and proposed a method called Fast R-CNN [5] to detect objects rapidly. This method reported higher detection accuracy and performed training in a single stage using a multi-task loss. Disk storage was also not required as it was in the case of R-CNN. Shaoqing Ren et al. [6] further improved Girshick's work and proposed Faster R-CNN and reduced the computation time. They also combined Fast R-CNN and RPN by sharing their convolutional features into a single network. This method outperformed both R-CNN and Fast R-CNN on publicly available image datasets. Joseph Redmon et al. [7] also presented YOLO, which could detects objects in one evaluation of CNN. It resizes the images to 448 x 448 and executes a single pass of CNN on the image to detect the objects and outperformed R-CNN. However, all these works have been proposed to perform detection and classification tasks on still images. It is more likely that leveraging these methods for videos can be limited in scope because the objects cannot be in the good position in all video frames. Especially under uncontrolled and complex conditions, where we can have blurring of objects, varying poses and illumination conditions, existing approaches

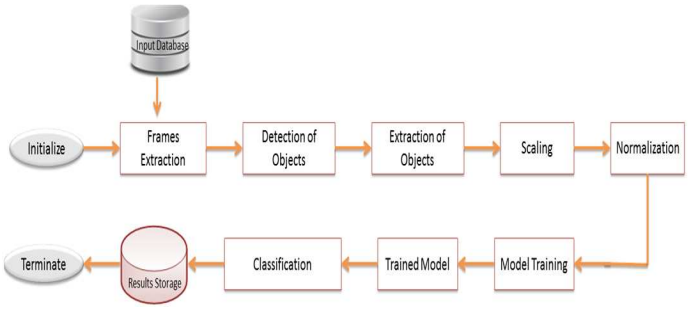

Fig. 3: Workflow of the Proposed Network

provide false positive detections, lack accuracy, and show less resilience to these changing conditions.

Limited recent work has investigated video classification for multimedia data using CNN. Approaches that do exist lack automation and require human assistance to perform object classification. The investigation of behavior that how it impacts hyper-parameter selection is scarce in recent literature. We provide an analysis of these parameters and present the optimal tuning parameters.

Also, existing deep learning-based methods use CNNs with some tweaked algorithms to perform multi-object detection/classification, leading to their own limitations. Approaches for multiple object classification in an image often use object detection algorithms and CNN is executed on top of these – acting as a means to aggregate multiple filters. A sliding window is often used to determine where the objects of interest are in the form of a bounding box, which contain the object of interest. A CNN is executed on top of these object detectors and all the bounding boxes are passed to the CNN to do object classification. This is extremely slow to run on large sized images or video frames. Even algorithms such as selective search, and faster variants are slow for video sequences. R-CNN [3] generally proposes about 2k regions and on each of these regions a CNN will run and extract high level features to do classification. These region-based proposals also tend to be extremely slow to execute. Also, multi-stage training is required to run the model.

We perform multi-object detection (faces of different individuals) through a Haar cascade classifier. Detected objects are treated as independent objects after extracting them from video frames. The problem considered in this manuscript is to process a large number of video streams in order to locate objects of interest. We are therefore dealing with different individuals captured at different locations, across different intervals of time within a large number of video streams. We therefore do not have intra-class variation (as we have same class in terms of persons) but we have inter-class variation (different individuals) in our dataset.

III. VIDEO ANALYSIS MODEL

We present a system using CNN to perform automatic object classification. We present our approach in this section and represent the system using a mathematical model. The mathematical modeling of the system aids in tuning and training of the system.

The proposed system for video analytics is based on decoded video streams. Initially all the video streams are encoded with H.264 encoding scheme to minimize the storage space capacity. The video streams are decoded to split them in video frames. For a stream of 120 seconds length, 3000 video frames will be generated. The analysis is performed on these frames. This approach enables the independent analysis of video frames from each other and leads to high throughput and scalability on the cloud resources. The training set is given by;

$$\text{“}Training\ DataSst\ X = x1, x2, \ldots xn\text{”} \tag{1}$$

where x1, x2 ... are decoded frames. The detection of desired object from whole frame and its extraction through cropping is an important preprocessing step for video analysis as shown in figure **??**. This shortens the processing by eradicating those areas from frames which do not contain objects. Haar cascade classifier [8] is used for the detection of objects from video frames. The haar cascade classifier uses haar features which are generated from the objects in video frames to perform detection. The detected objects are then extracted from frames. These extracted objects are fed into the processing pipeline of the deep network to perform object classification. A labeled frame is given as (x; c). The region of interest is represented as;

$$\text{“}R(x_0, y_0\ x_n, y_n)\text{”} \tag{2}$$

We extract the detected object patch which includes the surroundings of the object. Each video frame is scaled at a size of 150×150. This size has been selected according to the hyper-parameter tuning of the deep network based on the experimentation. The objects are further normalized as the deep networks works better when input is provided in normalized form. It is also to be noted that during the decoding and detection step, only those video frames are retained which contained the objects in them. All the video frames which do not possess any object are discarded. The normalized extracted objects are given as:

$$\text{“}X_{norm} = f(K(x);\ K(y))|(x;\ y)\text{”} \tag{3}$$

We have performed transformations including translation and skew to increase the training data. The greater the training data, the more will be the accuracy of the classifier. This technique proved to be very effective in feature learning algorithms since the classifier is exposed to much more training data with a variety of transformations[27][28][29][15]. This approach also reduces over-fitting and helps improving accuracy of the trained classifier.

Another advantage of applying these transformations is that they make the classifier invariant to typical transformations in the target object. These transformations in the target object can present themselves as serious challenges during object classification process and can drop the accuracy. So there is no need to handle these challenges separately as done in many

previous works [9][10][11]. However, it should be noted that the classifier will only be invariant to those variations in the target object on which it has been trained. Handling all of them such as occlusion is out of the scope of this paper.

Let 'T' denotes the transformations then the training dataset is given by;

$$\text{“}TX_{norm} = TX_{norm1}, TX_{norm2}, \ldots TX_{normn}\text{”} \quad (4)$$

Now when we have the dataset generated, we train the convolutional neural network. The convolutional and sub-sampling layers of the convolutional neural network are represented as;

$$\text{“}Conv_k, p = g(x_k, p * W_k, p + B_k, p)\text{”} \quad (5)$$

$$\text{“}Sub_k, p = g(\downarrow x_k, p * w_k, p + b_k, p)\text{”} \quad (6)$$

here g(.) is the ReLU activation function. Weights are represented by 'W'and biases are represented by 'b' respectively. ' * ' represents the two dimensional convolution operation. The inputs are downsampled in case of sub-sampling layer. The output from each layer represents a feature map. Multiple feature maps are extracted from each layer which is helpful in detecting multiple features of objects such as lines, edges and contours.

Instead of using the standard hyperbolic tangent non-linearity, we adopted 'ReLU' as suggested by [12][16]. ReLU is much more appropriate than tanh especially in case of bigger datasets as the network trains much faster. Traditional hyperbolic tangent non-linearity does not allow training the system on bigger datasets. The ReLU function has a range of [0,infinity], so it has the capability to model positive real numbers. The advantage of using ReLU is that it does not vanish as the value of 'x' increases as compared to sigmodal function. The max function is;

$$\text{“}1 \; if \; x > 0; \; 0 \; if \; x < 0\text{”} \quad (7)$$

In order to aid generalization we adopted Local Response Normalization. This normalization scheme mimics the behavior of real neurons and creates a competition amongst neuron outputs for big activities. Max pooling is used to perform sample based discretization or downsampling of an input representation (feature maps from convolutional layer in our case). Max pooling reduces the dimensionality, decreases the amount of parameters to learn and reduces the overall cost.

L2 regularization has been added to reduce over-fitting. It tries to penalize network weights that are large. It is given by;

$$\text{“}\lambda_2 \sum_i \theta_i^2 \quad (8)$$

where theta represents the network weights and lambda is lagrange multiplier which decides how significant this regularization should considered to be.

The deltas for the layers are;

$$\text{“}\triangle W_t, l = LearningRate \sum_{i=1}^{F} (x_i * D_i^h) + mn \triangle W_{(t-1,l)}\text{”} \quad (9)$$

Similarly;

$$\text{“}\triangle B_t, l = LearningRate \sum_{i=1}^{F} D_i^h + mn \triangle B_{(t-1,l)}\text{”} \quad (10)$$

Similarly;

$$\text{“}\triangle W_t, l = LearningRate \sum_{i=1}^{F} (\downarrow x_i * D_i^h) + mn \triangle W_{(t-1,l)}\text{”} \quad (11)$$

Also;

$$\text{“}\triangle b_t, l = LearningRate \sum_{i=1}^{F} D_i^h + mn \triangle b_{(t-1,l)}\text{”} \quad (12)$$

The loss function is;

$$\text{“}L(x) = LearningRate \sum_{x_i -> X} \sum_{x_i -> T_i} l(i, x_i T)\text{”} \quad (13)$$

here l(i,xT) is loss function for convolutional neural network that we are trying to minimize.

SGD is represented as;

$$\text{“}W_{t+1} = W_t - \alpha \delta L(\theta_t)\text{”} \quad (14)$$

The momentum term is represented as;

$$\text{“}V_{t+1} = \rho v_t - \alpha \delta L(\theta_t)\text{”} \quad (15)$$

$$\text{“}W_{t+1} = W_t + V_{t+1}\text{”} \quad (16)$$

The softmax layer is given as;

$$\text{“}l(i, x_i T) = M(e_i, f(x_i T))\text{”} \quad (17)$$

IV. Architecture and Implementation

The proposed video analysis approach is compute intensive and operates on large datasets. We have tackled this problem by optimizing the code, tuning the hyper-parameters properly and introducing parallelism [33] by using spark. Parallelism is achieved by distributing the dataset into small subsets and then passing over these subsets of data to separate neural network models as shown in figure 4. The models are trained in parallel and the resultant parameters for each model are then iteratively averaged and collected at the master node. This approach helped in speeding up the network training even on larger datasets.

The training process starts by first loading the training dataset into the memory. The master node which also acts as the spark driver loads the initial parameters and the network configuration. The network configuration of our spark cluster and deep learning model is shown in Table 1: The dataset is partitioned in a number of subsets. This division is dependent on the configuration of the training master. These subsets of data are distributed to various workers along with the configuration parameters. Each worker then performs training on its allocated dataset. Once the training by all the workers is completed, the results are averaged and returned to master which has a fully trained network which is used for classification.

The master node of spark loads the initial network configuration and parameters. The master is termed as driver node

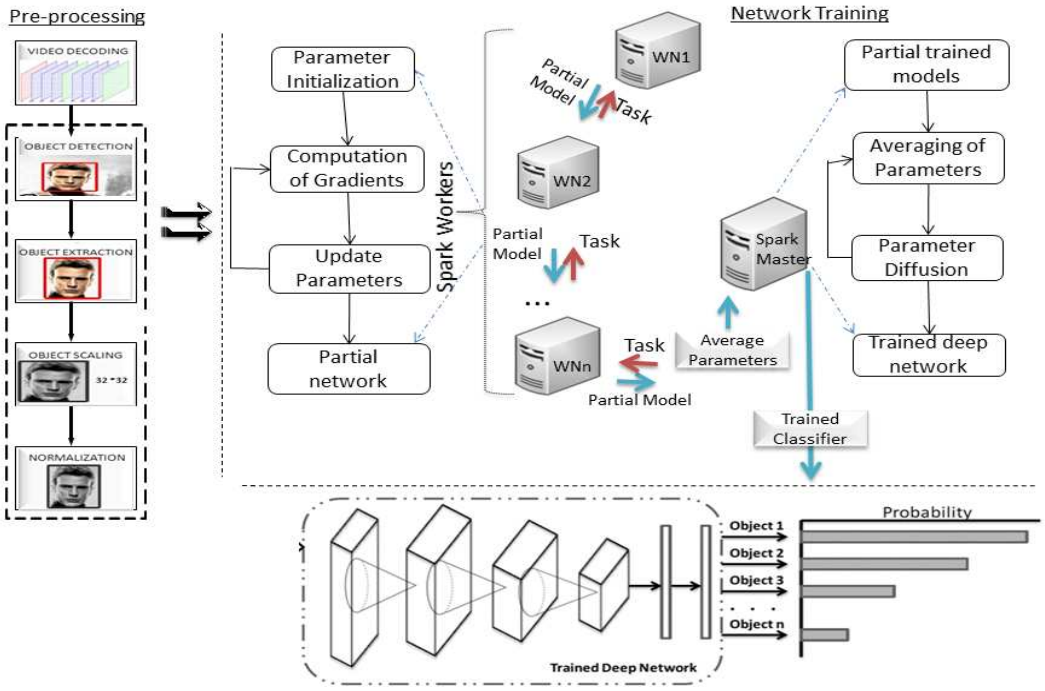

Fig. 4: Architecture of Distributed Cluster

TABLE I: Configuration of Spark Cluster and Deep Network

Spark.driver.cores	4	No. of layers	8
Spark.driver.memory	8GB	Average Rate	1
Spark.executor.memory	8GB	Optimization	GD
Spark.executor.cores	4	Activation	ReLu
Spark.memory.fraction	4	LearningRate	0.0001
Spark.serializer	kryo	Regularization	L2

as well because it is responsible to drive other nodes of the cluster by distributing parameters among them. It also contains the knowledge that how data is to be divided. On the basis of data division parameter, the dataset is partitioned into the subsets. These subsets along with the configuration parameters are then distributed among worker nodes. Each worker works on a partial model and the results are averaged together with the help of iterative averaging. The master node then contains the trained classifier.

The separation of training data into subsets and then training the model with these subsets of data by averaging parameters is a feasible approach for our system because we operate with limited worker nodes in our cloud and the parameters for estimation are also small. We use the same model for each worker node but train them on different data shards (mini-batches). We then obtain the gradient for each split of the mini-batch from each model and compute the overall average using parameter averaging. This technique works faster for small networks as in our proposed system and is ideal for scenarios involving matrix computations which happens quite often in convolutional neural networks.

The compute cluster consists of one master node and eight worker nodes. The averaging frequency is set to 1 for all the experiments. The dataset comprising the size of 100GB is divided into various subsets of data. Each subset is further divided into various minibatches depending upon the configuration. Training is performed on each subset by allocating each minibatch to each worker. Since the dataset is large in size it was not possible to load the whole dataset into memory at once. So we have first exported the minibatches of datasets to disk (HDFS) known as dataset objects. The datasets are exported in batched and serialized form. We have used kryo serialization to perform serialization of our dataset. This approach of saving the dataset to disk is much more efficient and faster as compared to loading the whole dataset in memory. This approach consumes less memory and reduces split overhead. The dataset object has a number of examples based on the size of dataset object. Kryo serialization takes least amount of time to serialize objects and improves performance. It can serialize objects much quickly and efficiently and offers more compact serialization than Java. The serialization framework provided by java has high CPU and RAM consumption which makes it inefficient for large scale data objects.

This is also quiet important to set the rate of parameter averaging. If this is too low, this will create overhead in parameter initialization and will cause delay in network communication. Similarly, if it is high, it will degrade the performance. In the proposed video analytics system, the good performance is obtained with 16 mini-batches. These mini-batches are started in an asynchronous fashion which reduces the delay. The data

repartitioning is also a critical parameter to be defined. It defines when data is to be repartitioned and plays an important role in utilizing all the resources of the cluster efficiently. A value of 0.6 is chosen for this.

The locality configuration is also defined as the proposed algorithm has high demand of computation, so single task per executor is executed. It is therefore much suitable to shift data to executor which is free. The default configuration of spark waits for a free executor. This requires the data to be copied across the network. Another important note is that we have avoided the allocation of memory on JVM heap space by passing pointers for various numerical tasks. It is not required to load the data from JVM heap to execute operations on it; neither has it required data transmission (processed results) back to JVM. This helps to avoid the data transfer time and a decrease in overall execution time of the system. This also avoids memory overhead required for each task.

We have employed iterative mapreduce instead of simple mapreduce for our proposed application. Iterative mapreduce is an advanced form of mapreduce in which multiple passes of the mapreduce operation are performed. Single pass does quite well for the application which are not iterative. As our application is built upon deep learning algorithm, it is highly iterative and makes full use of the iterative map-reduce. A sequence of map-reduce operations are performed in which each mapreduce operation is performed in cascaded fashion.

In the implementation phase, the video dataset is first loaded into the memory. It is preprocessed so that it can be further used for training the deep multilayer network. The preprocessing starts with frame decoding using FFMPEG library [13]. The objects of interest are then detected and extracted from the video frames by using haar cascade classifier. Haar cascade classifier is built on top of haar features which are generated from the objects in video frames to perform detection.

The objects which are extracted necessitate the use of N-dimensional arrays which could hold the pixel values. We have made the use of nd4j for java [14]. It consumes minimum memory and supports fast numerical computing for java. The loading of data into the memory and training of the network are handled by two separate processes. This makes the data loading process simple and is supported by the nd4j library. The data after loading into the memory is normalized. This normalization of data helps to train the neural network properly as it is based upon gradient descent optimization approach for network training. The gradient descent approach having their activation functions in this range helps to improve the performance.

A dataset iterator is defined to iterate over the data present in the memory. The iterator fetches the data from memory in a vectorised format. The iterator moves on to the dataset objects which contains multiple training examples along with their labels. An n-dimensional array is created to store examples and labels. The high volumes of data makes it infeasible to load the data into the memory at once. So many minibatches are created. These minibatches help to tackle the memory requirements problem. A value of 12 for the minibatch is used in our system.

The value of learning rate has been selected to be 0.0001. We have selected this value carefully on the basis of experimentation. We observed during the experiments that a high value of learning rate can cause divergence and the divergence can stop the learning. On the other hand, setting learning rate to a small value causes slow convergence.

V. Experimental Setup

The details of our experimental setup which is utilized to implement the system is presented in this section. The main focus of the results generated by using this experimental setup is accuracy of the proposed algorithm, scalability, precision and performance of the system. The accuracy of the system is measured by precision, Recall and F1 score. The scalability and performance is demonstrated by analyzing aspects of the system including transfer time of data to cloud node and the overall analysis time.

The proposed architecture for analysing video streams consists of cloud resources. The compute nodes have multi-cores for processing in which most of the video analytics operations are performed. In order to execute the experiments, we constructed a cluster of eight nodes on the cloud infrastructure. The multiple instances running on the cloud have OpenStack [22] with ubuntu version of 15.04. This cluster is used to deploy and evaluate the proposed system. The configuration of the cluster is as follows: Each node in the cluster possesses a secondary storage of 100 GB. There are 4 VCPUs running at a frequency of 2.4 GHz. The total main memory has a size of 16 GB. The results generated by these experiments will help to deploy the system on a much bigger infrastructure as per requirements of an application.

The video dataset which is used to train and test the system is generated in a constrained environment. The streams are captured with individuals facing towards the camera. However, it also contains frames which have individuals with side, front and rear pose. Most of the video streams do not pose illumination or other challenges. The test dataset comprises of 88,432 video frames.

The input video streams are H.264 format encoded. The frame rate for each video stream in our database is 25 fps. The data rate is 421 kbps and the bit rate of video streams is 461 kbps respectively. These video streams are decoded to produce separate video frames. The video stream of one minute of length generates a decoded frame set of 1500 frames. The data size of each video frame is 371 kb.

"Apache Spark" is adopted for parallel and distributed training of the deep network. The video dataset is loaded in spark which executes executors to perform the network training. The dataset objects are used by the executors to execute training of the network. The iterative MapReduce framework utilized in this work executes multiple analysis tasks. These analysis tasks are executed in multiple stages. The analysis tasks are rescheduled if a task failure occurs.

The spark context is utilized to load the video dataset and is then stored into multi-dimensional arrays. The multi-dimensional arrays represents the data in the form of tensors which are then passed through multiple layers for training. The starting layer of convolutional neural network has a dimension

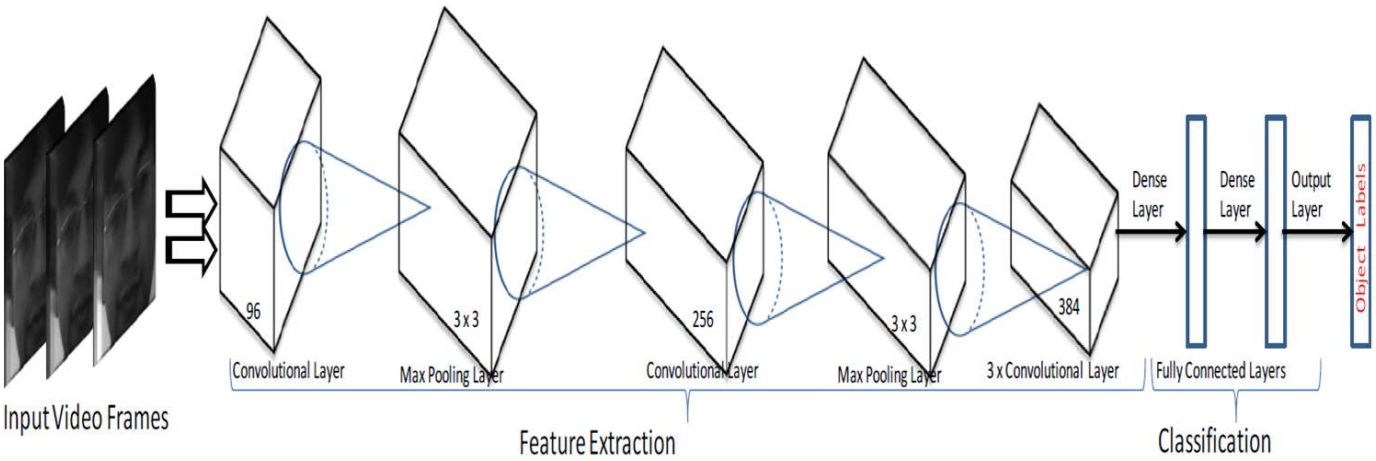

Fig. 5: Schematic Diagram of the Proposed Network

of 150 x 150 x 1. It has 96 kernels in it. The stride of the kernels is set to be 4 x 4 with the kernel size of 11 x 11 x 1. The layer following the first convolutional layer has 256 kernels in it with a stride of 2 and has a size of 1 x 1. The remaining layers has a total of 284 kernels in them. These convolutional layers operate on nonZeroBias.

There is a max-pooling layer next to the convolutional layers with a size of 3 x 3 as shown in figure 5. The convolutional layers and pooling layer are followed by the fully connected layers. The fully connected layers have a total of 4096 neurons in them. The kernels and neurons of the subsequent layers has a connection with the previous layers. We have also added local response normalization layers, max pooling layers and added ReLU as non-linearity layer.

VI. Experimental Results

In this section we present the results of the proposed system using the experimental setup detailed in section V. We first analyze the results generated by tuning the hyper-parameters of deep model to various values and propose the parameters which could potentially produce best results.The trained system on the proposed parameters is then evaluated with different performance characterization including accuracy, scalability and performance of the system. The Precision, Recall and F1 score are also considered as the performance characterization. The scalability of the system is analysed by measuring the time to transfer data to cloud and overall time of analysis of data. The results from the object classification pipeline are presented at the end of the section.

A. Hyper-parameter Tuning

There are a number of parameters which can be tracked during the training of a deep network. These parameters provide intuitions about the settings of different hyper-parameters and help to make a decision that whether the setting should be changed in order to have more efficient learning. The parameters are tracked and represented in the form of graphs over multiple time stamps in order to observe the trend in the behavior of the system. The x-axis of the plot in figure

Fig. 6: Model Scores

6 represents iterations and the number of iterations depends on the settings of batch size. While the loss function value $L(x) = LR \sum_{x_i->X} \sum_{x_i->T_i} l(i, x_i T)$ of current mini-batch is depicted on the y-axis of the plot in figure 6. The loss function value is evaluated during the forward pass of the back-propagation on the individual batches. The grey line in the graph represents the running average of the loss on each iteration. It gives a better visualization to analyze the trend in the graph of the loss function. The graph depicts that the learning rate is tuned properly as a decreasing trend in the graph is observed after each iteration over time. We kept on changing the learning rate unless score became stable. The learning rate has been varied to many different values and three of them le-2, le-4 and le-6 are shown in the graph. Le-2 proved to be good for the divergence of learning curve as shown in figure 6.

The proper normalization of the data is a major factor in the divergence of the learning curve. It is also an indication that the L2 employed with SGD "$W_{t+1} = W_t - \alpha \delta L(\theta_t)$" is good adopted scheme. Here "α" varied to le-2, le-4 and le-6. The initialization of weights has been made random. The bottom two graphs with a learning rate of le-4 and le-6 remained

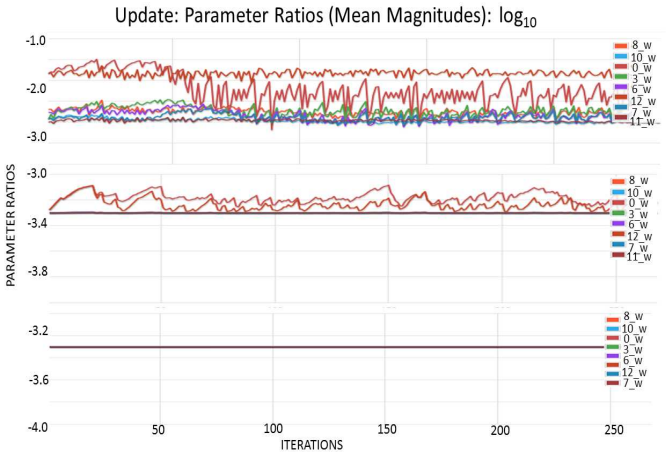

Fig. 7: Parameter Ratios

Fig. 9: Model Scores at Various Iterations

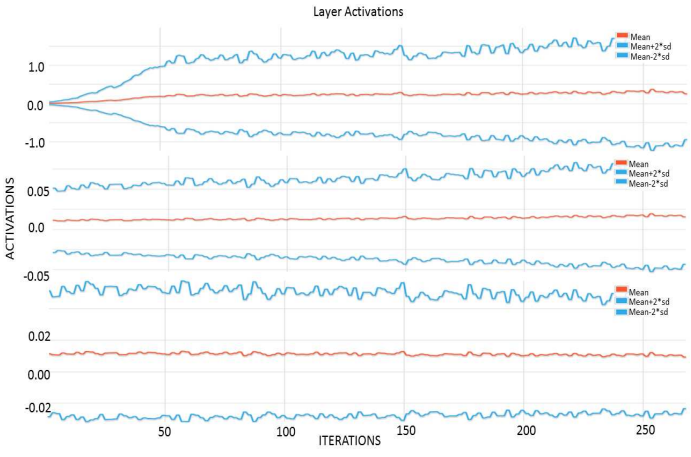

Fig. 8: Layer Activations

unable to show the decreasing trend and followed a stable state over multiple iterations. Both the graphs remained above 1.0 on y-axis.

Another important parameter which can be used to track in order to have an intuition about the efficient learning of the system is the ratio of weights (updates). It is not beneficial to track the raw gradients but the updates of the weights. It can also be helpful to track this ratio for every set of parameters. The parameter ratios are depicted in figure 7. The trend in the graph indicates selection of a good learning rate and proper initialization of network hyper-parameters. The parameter weights are represented by different colored lines in the graph. A high divergence of the parameters from -3 on a log10 chart indicates that the parameters are not properly initialized.

The layer activations of first layer utilized in our system are depicted in figure 8. A stability in the layer activations graph can be observed clearly which shows that the network is stable. It is also an indication of the proper initialization of weights of the layers. The regularization scheme i.e. $\lambda_2 \sum_i \theta_i^2$ is well adopted. The convergence of ratio as seen in the graph shows that the parameters are initialized correctly and are well selected. The other lower graphs of figure 11 with lambda values do not show a stability trend.

B. Training on Tuned parameter values

We have trained the system on the proposed hyper-parameters for our video object classification pipeline and evaluated the performance. Figure 9 shows the value of loss function at various iterations on the current minibatch. The graph is drawn against training scores of the network and training iterations. It can be seen that the graph converges which shows that the learning rate LR = 0.0001 is a well selected learning rate. The decreasing trend of the graph is also an indication that "L2 normalization scheme $\lambda_2 \sum_i \theta_i^2$" with "SGD $W_{t+1} = W_t - \alpha \delta L(\theta_t)$" is a good approach for the training of our network. A bit of a noise in the graph is observed but it is very low variation in a small range and is not an indicative of poor convergence of learning.

Figure 10(a), figure 10(b) and figure 10(c) show the standard deviations of layer activations, gradients and updates of parameters. A stable trend is observed in this graph which shows that the system is capable of coping with the problem of vanishing or exploding activations. It also shows that the weights of the layers have been well selected and regularization scheme is properly adopted.

The histogram of layer parameters and layer updates are depicted in Figure 11 Figure 12 respectively. The normalized "Gaussian distribution" can be seen in graphs. It shows that the weights are properly initialized with sufficient regularization present in the system. The layer updates graph also shows that the system is not exposed to vanishing gradient because of the utilization of non-linearity $h = max(0, a)$.

Figure 13(a), figure 13(b) and figure 13(c) show the standard deviations of layer activations, gradients and updates of parameters for the first convolution layer of the network. The proposed system made use of off heap memory and most of the memory is not allocated on the JVM heap but outside of the JVM. This helps to perform the numerical operations faster as data needs not to be copied to and from the JVM but pointers can be passed around for numerical computations avoiding data copying issue.

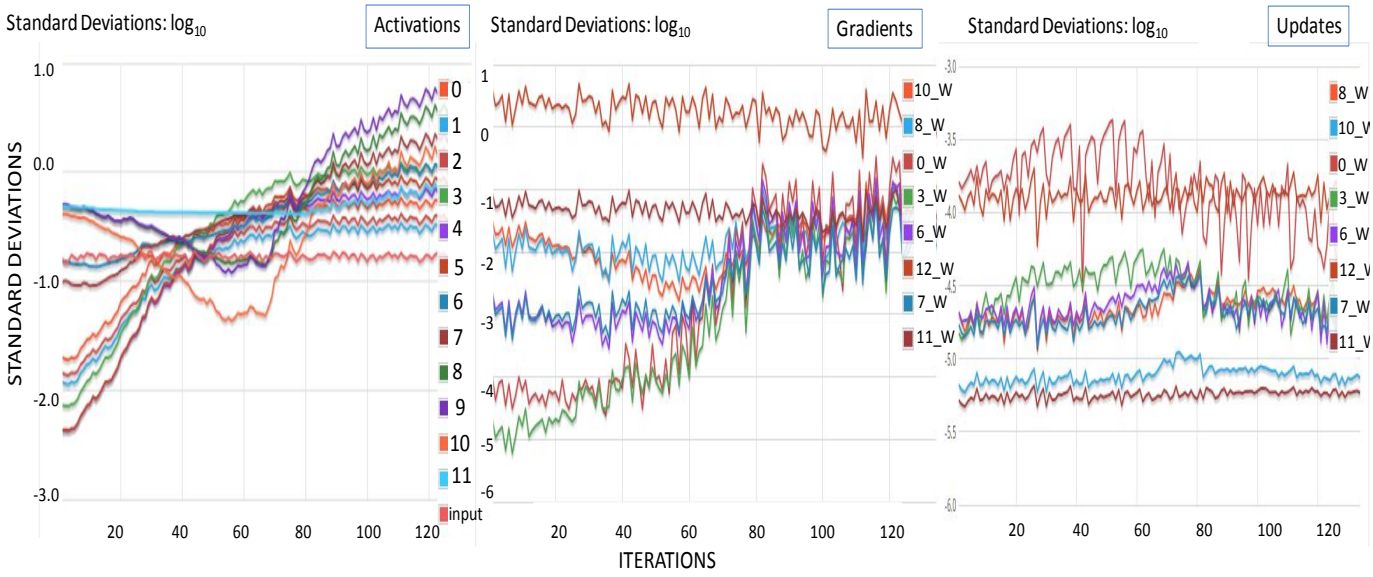

Fig. 10: Standard Deviations of activations and Parameter Updates

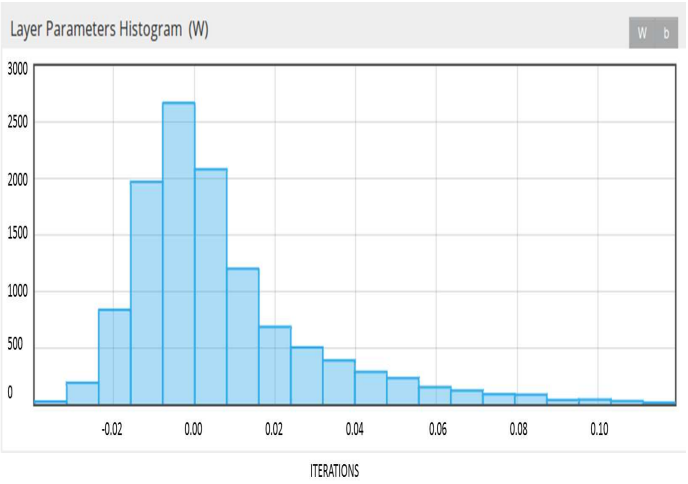

Fig. 11: Histogram of Layer Parameters

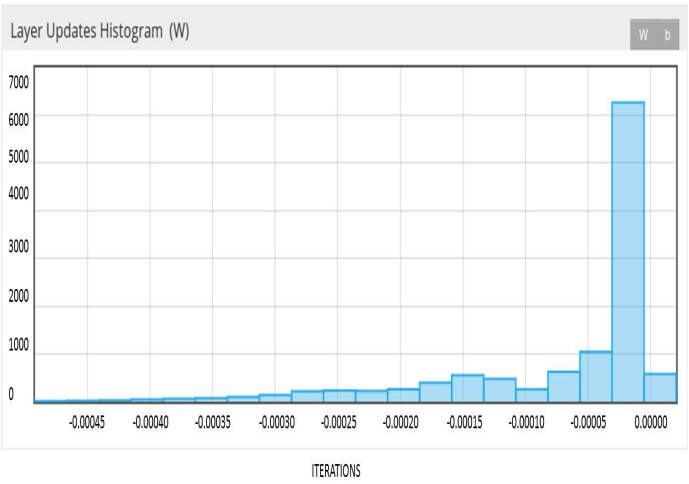

Fig. 12: Histogram of Layer Updates

C. Performance Characterization and Scalability of the System

The accuracy of the proposed system is measured by the following performance characterization: recall, precision (positive prediction value) and F1 score. The test dataset comprises of 88,432 video frames in total. The precision is turned out to be 0.9708. The recall of the system is recorded to be 0.9636. And the F1 score is found to be 0.9672. The recall and the F1 score are calculated by the following equations:

$$''Recall = TP/(TP + FN)'' \tag{18}$$

$$''F1 = 2TP/(2TP + FP + FN)'' \tag{19}$$

It was observed from the results that there is also some miss-classification of the video frames as well. Few objects are recorded as false positives in the system. There can be number of things which could be the reason for the miss-classification. Some miss-classifications could be due to the variance in the pose, illumination conditions and blur effects. As the training of the classifier was performed on the dataset which was captured under strict controlled conditions, high variance could lead to the miss-classification of various subjects.

The scalability is tested by executing it on distributed infrastructure over multiple nodes. The system is evaluated mainly on the following parameters: i) transfer time of data to cloud nodes ii) total time of analysis iii) analysis time with varying dataset sizes. Spark executes many executors and these executors accesses a RDD object in each iteration. Spark has a cache manager which handles the iterations outcomes in memory. If the data is not required anymore, it is stored on disk.

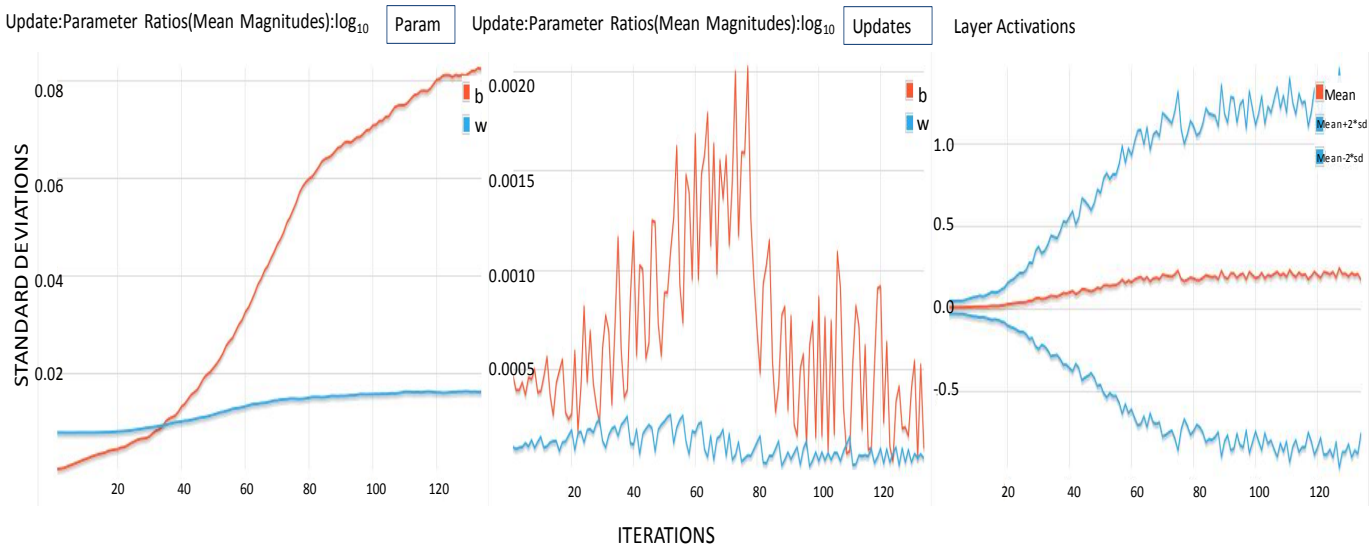

Fig. 13: Standard Deviations for First Layer

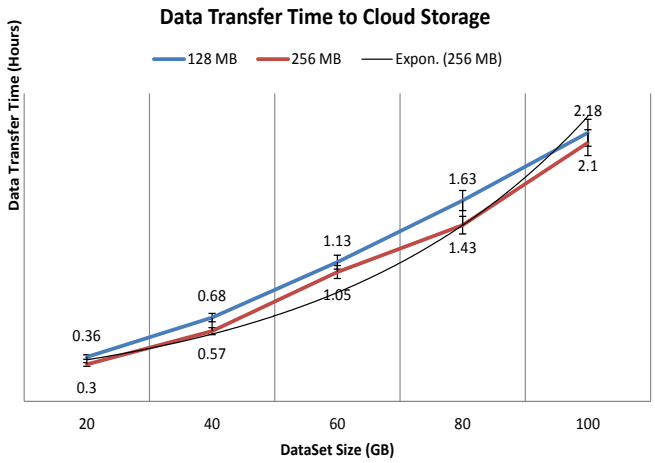

Fig. 14: Total Transfer Time

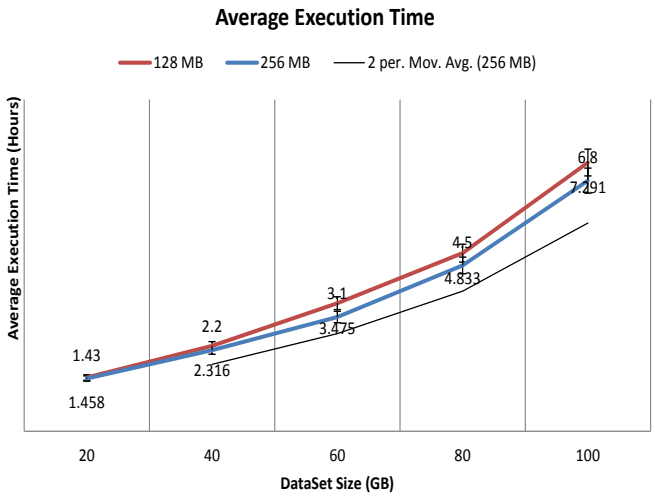

Fig. 15: Average Time

Each video stream in our database has a frame per second rate of 25. These videos are decoded to produce separate video frames. The total number of decoded video frames is directly proportional to the duration of video stream being analyzed. The video stream of one minute of length generates a decoded dataset of 1500 frames.

The size of the input dataset varies from five gigabytes to hundred gigabytes. The large number of frames are bundled with the help of a batch process. The bundled frames are shifted to cloud infrastructure for processing. The time required to bundle the frames is proportional to the input video frames size. The dataset ranging from ten gigabytes to hundred gigabytes requires a bundling time of 0.25 to 3.8 hours. Inclusion of larger data increases the data bundling time.

The bundled video frames are then transferred to cloud for analysis. There are many factors on which the transfer time to cloud depends including bandwidth of the network, block size and also the amount of data which is to be transferred. An estimated transfer time for different data sizes as shown in Figure 14. It was observed that the transfer time for a dataset size of twenty gigabytes to hundred gigabytes varied from 0.36 to 2.18 hours . The data transfer time has also been measured with a block size of 256MB and its effect is shown in figure 14. In order to measure the time of network training, multiple tests are carried out on multiple dataset sizes. We have then calculated the average execution time of each dataset size and plotted in Figure 15. It was seen from the results that the increase in dataset size directly increases time of execution.

D. Video Object Classification Pipeline

The trained classifier from cloud is saved locally and is further used to locate the objects of interest. The target object which is to be located from the video streams is passed through the trained classifier to perform classification. The target object which is to be classified is passed through the same

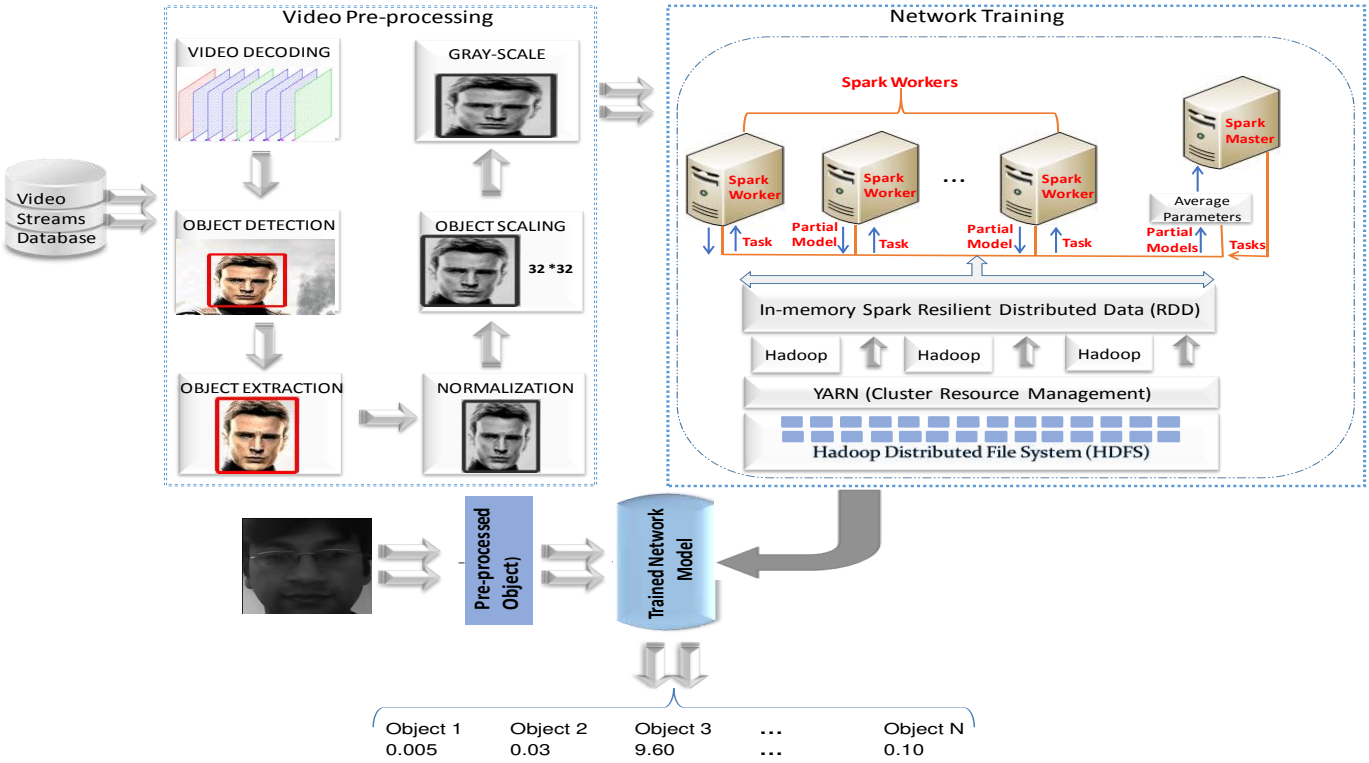

Fig. 16: Video Object Classification Pipeline

preprocessing steps to make it appropriate for the classifier. It is also scaled and normalized to make it appropriate for the classifier.

The classifier returns the probabilities of the possible labels but not the labels itself. The labels of all the objects present in all the video streams were already stored in the database beforehand. The classification process ends up in generating the probabilities of the matched objects. The object with the highest probability indicates the classification of the desired object which was being searched from the video streams. Very low probabilities against all the objects indicate that the target object is not present in all of the video streams present in the database.Figure 16 depicts the phenomenon of object classification.

Figure 17 depicts the probabilities of some of the objects generated by the classifier. The marked objects which were fed into the trained network are listed on the right hand side of the graph. We have shown results from 8 different objects for this set of experiments. The probabilities generated by the classifier against each object are shown in different columns of the table. The probabilities near to 1 depict a closer match of marked object and the probabilities close to 0 depicts the unavailability of objects in the video stream database.

It can be seen that the trained classifier generates a high probability against the marked object if its training instances are present in the database. The other labels of objects come up with a low probability value. Figure 17 depicts the graphical representation of classification procedure. The 10 experiments are represented on each index of the x-axis. Different probabilities generated by each experiment are represented on y-axis of the graph.

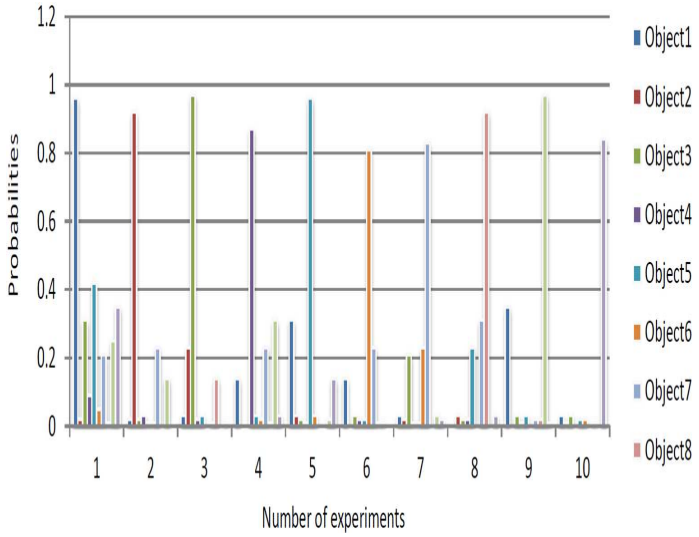

Fig. 17: Classification of Marked Object

VII. Conclusion and Future Work

An object classification system is developed and presented. The system is built upon deep convolutional neural network to perform object classification. The system learns different features from many video streams and performs training on an in-memory cluster. This makes the system more robust to classification errors by rapidly incorporating diverse features from training dataset.

The system is validated with the help of a case-study using real-life scenario. Numerous experiments on the testing dataset proved that the system is accurate with an accuracy of 0.97 as well as precise with a precision of 0.96 respectively. The system is also capable of coping with varying number of nodes or increased volumes of data. The time required to analyse the video data depicted an increasing trend with the increasing amount of video data to be analysed in the cloud. The analysis time is directly reliant on the amount of data being analyzed. The sheer volumes of video data necessitate additional time to carry out object classification. The analysis time can be decreased with the addition of nodes in the infrastructure.

We would like to leverage and optimize other deep learning models in future including reinforcement learning based methods. The reinforcement learning will help to classify other objects as well like vehicles without necessitating any metric learning stage. The development of a toolkit which could automate the process of hyper-parameter optimization will be included as part of the improvements in the system.

We also intend to develop a rule based recommendation system for cloud based video analytics which will provide recommendations for hyper-parameter tuning on the basis of input dataset and its characteristics. It will also take into account the configurations of underlying in-memory compute cluster and will suggest appropriate tuning parameters for both deep learning model and in-memory cluster.

REFERENCES

[1] Anjum, A., Abdullah, T., Tariq, M., Baltaci, Y. and Antonopoulos, N. Video stream analysis in clouds: An object detection and classification framework for high performance video analytics. IEEE Transactions on Cloud Computing. 1(1):1–14, 2016

[2] J. Bergstra, D. Yamins, and D.D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Proc. of ICML, pages 115–123, 2013.

[3] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1-9).

[4] Girshick, R., Iandola, F., Darrell, T. and Malik, J., 2015. Deformable part models are convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 437-446).

[5] R. Girshick, "Fast R-CNN", ICCV 2015

[6] S. Ren, K. He, R. Girshick, J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", IEEE Transactions on Pattern Analysis and Machine Intelligence. 2016.

[7] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection". 2016, pp. 779-788

[8] Lienhart, R. and Maydt, J., 2002. An extended set of haar-like features for rapid object detection. In Image Processing. 2002. Proceedings. 2002 International Conference on (Vol. 1, pp. I-I). IEEE.

[9] Erhan, D., Szegedy, C., Toshev, A. and Anguelov, D., 2014. Scalable object detection using deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2147-2154).

[10] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[11] Tang, J., Deng, C. and Huang, G.B., 2016. Extreme learning machine for multilayer perceptron. IEEE transactions on neural networks and learning systems, 27(4), pp.809-821.

[12] Dahl, G.E., Sainath, T.N. and Hinton, G.E., 2013, May. Improving deep neural networks for LVCSR using rectified linear units and dropout. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on (pp. 8609-8613). IEEE.

[13] https://ffmpeg.org/ Last Accessed [03/01/2018]

[14] WWW.nd4j.org/ Last Accessed [03/01/2018]

[15] Yaseen, M.U., Anjum, A. and Antonopoulos, N., 2017. Modeling and Analysis of a Deep Learning Pipeline for Cloud based Video Analytics. In Proceedings of the fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies. ACM.

[16] G. Dahl, T. Sainath, and G. Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In Proc. of ICASSP, pages 8609–8613. IEEE, 2013.

[17] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In Proc. of NIPS, pages 2546–2554, 2011.

[18] Q. Wang, J. Gao and Y. Yuan, "Embedding Structured Contour and Location Prior in Siamesed Fully Convolutional Networks for Road Detection," in IEEE Transactions on Intelligent Transportation Systems.2017, DOI: 10.1109/TITS.2017.2749964

[19] Y. Yuan, Y. Lu and Q. Wang, "Tracking as a Whole: Multi-Target Tracking by Modeling Group Behavior With Sequential Detection," in IEEE Transactions on Intelligent Transportation Systems.2017, DOI: 10.1109/TITS.2017.2686871

[20] Bergstra, J., Yamins, D. and Cox, D.D., 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In Proceedings of the 12th Python in Science Conference (pp. 13-20).

[21] Zhang, Q., Liu, W., Tsang, E. and Virginas, B., 2010. Expensive multiobjective optimization by MOEA/D with Gaussian process model. IEEE Transactions on Evolutionary Computation, 14(3), pp.456-474.

[22] https://www.openstack.org/ Last Accessed [03/01/2018]

[23] Wang, N. and Yeung, D.Y., 2013. Learning a deep compact image representation for visual tracking. In Advances in neural information processing systems (pp. 809-817).

[24] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.A., 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research, 11(Dec), pp.3371-3408.

[25] Boureau, Y.L. and Cun, Y.L., 2008. Sparse feature learning for deep belief networks. In Advances in neural information processing systems (pp. 1185-1192).

[26] Lee, H., Pham, P., Largman, Y. and Ng, A.Y., 2009. Unsupervised feature learning for audio classification using convolutional deep belief networks. In Advances in neural information processing systems (pp. 1096-1104).

[27] Van Dyk, D.A. and Meng, X.L., 2001. The art of data augmentation. Journal of Computational and Graphical Statistics, 10(1), pp.1-50.

[28] Cui, X., Goel, V. and Kingsbury, B., 2015. Data augmentation for deep neural network acoustic modeling. IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP), 23(9), pp.1469-1477.

[29] Zhu, J., Chen, N., Perkins, H. and Zhang, B., 2014. Gibbs max-margin topic models with data augmentation. Journal of Machine Learning Research, 15(1), pp.1073-1110.

[30] Yaseen, M.U., Anjum, A. and Antonopoulos, N., 2016, December. Spatial frequency based video stream analysis for object classification and recognition in clouds. In Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (pp. 18-26). ACM.

[31] Yaseen, M.U., Anjum, A., Rana, O. and Hill, R., 2017. Cloud-based scalable object detection and classification in video streams. Future Generation Computer Systems. https://doi.org/10.1016/j.future.2017.02.003

[32] A. R. Zamani, M. Zou, J. Diaz-Montes, I. Petri, O. Rana, A. Anjum and M. Parashar, 2017, Deadline Constrained Video Analysis via In-Transit Computational Environments, IEEE Transactions on Services Computing. DOI: 10.1109/TSC.2017.2653116

[33] McClatchey, R., Anjum, A., Stockinger, H. et al. J Grid Computing (2007) 5: 43. https://doi.org/10.1007/s10723-006-9059-z